Using the 8XC751 microcontroller as an I²C bus master

AN422

DESCRIPTION

The 83C751/87C751 Microcontroller offers the advantages of the 80C51 architecture in a small package and at a low cost. It combines the benefits of a high-performance microcontroller with on-board hardware supporting the Inter-Integrated Circuit (I²C) bus interface.

The I²C bus, developed and patented by Philips, allows integrated circuits to communicate directly with each other via a simple bidirectional 2-wire bus. The comprehensive family of CMOS and bipolar ICs incorporating the on-chip I²C interface offers many advantages to designers of digital control for industrial, consumer and telecommunications equipment. A typical system configuration is shown in Figure 1.

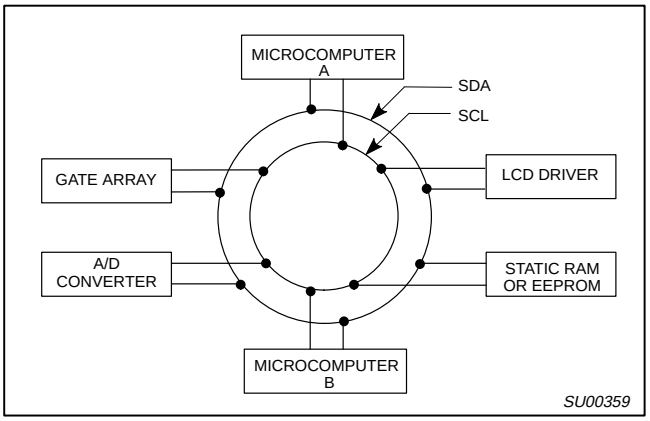

Figure 1. Typical I²C Bus Configuration

Interfacing the devices in an I²C based system is very simple because they connect directly to the two bus lines: a serial data line (SDA) and a serial clock line (SCL). System design can rapidly progress from block diagram to final schematic, as there is no need to design bus interfaces, and functional blocks on a block diagram correspond to actual ICs. A prototype system or a final product version can easily be modified or upgraded by 'clipping' or 'unclipping' ICs to or from the bus. The simplicity of designing with the I²C bus does not reduce its effectiveness; it is a reliable, multimaster bus with integrated addressing and data-transfer protocols (see Figure 2). In addition, the I²C-bus compatible ICs provide cost reduction benefits to equipment manufacturers, some of which are smaller IC packages and a minimization of PCB traces and glue logic.

The availability of microcontrollers like the 83C751, with on-board I²C interface, is a very powerful tool for system designers. The integrated protocols allow systems to be completely software defined. Software development time of different products can be reduced by assembling a library of reusable software modules. In addition, the multimaster capability allows rapid testing and alignment of end-products via external connections to an assembly-line computer.

The mask programmable 83C751 and its EPROM version, the 87C751, can operate as a master or a slave device on the I²C small area network. In addition to the efficient interface to the dedicated function ICs in the I²C family, the on-board interface facilities I/O and RAM expansion, access to EEPROM and processor-to-processor communications.

The multimaster capability of the I²C is very important but many designs do not require it. For many systems, it is sufficient that all communications between devices are initiated by a single, master processor. In this application note, use of the 8XC751 as an I²C bus master is described. Some of the technical features of the bus and the 83C751's special hardware associated with the I²C are discussed. Also included is a software example demonstrating I²C single master communications. Note that the sample routines are quite general, and therefore may be transferred easily to many applications.

The discussion of the I²C bus characteristics in this application note is by no means complete. Additional information for the I²C bus and the S83C751 Microcontroller can be found in the Microcontroller Users' Guide.

THE I²C BUS

The two lines of the I²C-bus are a serial data line (SDA) and a serial clock line (SCL). Both lines are connected to a positive supply via a pull-up resistor, and remain HIGH when the bus is not busy. Each device is recognized by a unique address—whether it is a microcomputer, LCD driver, memory or keyboard interface—and can operate as either a transmitter or receiver, depending on the function of the device. A device generating a message or data is a transmitter, and a device receiving the message or data is a receiver. Obviously, a passive function like an LCD driver could only be a receiver, while a microcontroller or a memory can both transmit and receive data.

Masters and Slaves

When a data transfer takes place on the bus, a device can either be a master or a slave. The device which initiates the transfer, and generates the clock signals for this transfer, is the master. At that time any device addressed is considered a slave. It is important to note that a master could either be a transmitter or a receiver; a master microcontroller may send data to a RAM acting as a transmitter, and then interrogate the RAM for its contents acting as a receiver—in both cases performing as the master initiating the transfer. In the same manner, a slave could be both a receiver and a transmitter.

The I²C is a multimaster bus. It is possible to have, in one system, more than one device capable of initiating transfers and controlling the bus (Figure 2). A microcontroller may act as a master for one transfer, and then be the slave for another transfer, initiated by another processor on the network. The master/slave relationships on the bus are not permanent, and may change on each transfer.

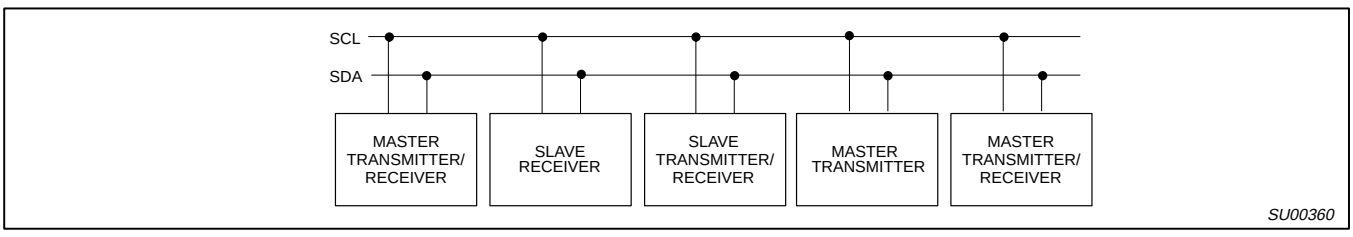

Figure 2. I²C Bus Connection

Using the 8XC751 microcontroller as an I²C bus master

AN422

As more than one master may be connected to the bus, it is possible that two devices will try to initiate a transfer at the same time. Obviously, in order to eliminate bus collisions and communications chaos, an arbitration procedure is necessary. The I²C design has an inherent arbitration and clock synchronization procedure relying on the wired-AND connection of the devices on the bus. In a typical multimaster system, a microcontroller program should allow it to gracefully switch between master and slave modes and preserve data integrity upon loss of arbitration. In this note, a simple case is presented describing the S83C751 operating as a single master on the bus.

Data Transfers

One data bit is transferred during each clock pulse (see Figure 3). The data on the SDA line must remain stable during the HIGH period of the clock pulse in order to be valid. Changes in the data line at this time will be interpreted as control signals. A HIGH-to-LOW transition of the data line (SDA) while the clock signal (SCL) is HIGH indicates a Start condition, and a LOW-to-HIGH transition of the SDA while SCL is HIGH defines a Stop condition (see Figure 4). The bus is considered to be busy after the Start condition and free again at a certain time interval after the Stop condition. The Start and Stop conditions are always generated by the master.

The number of data bytes transferred between the Start and Stop condition from transmitter to receiver is not limited. Each byte, which must be eight bits long, is transferred serially with the most significant bit first, and is followed by an acknowledge bit. (see Figure 5). The clock pulse related to the acknowledge bit is generated by the master. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, while the transmitting device releases the SDA line (HIGH) during this pulse (see Figure 6).

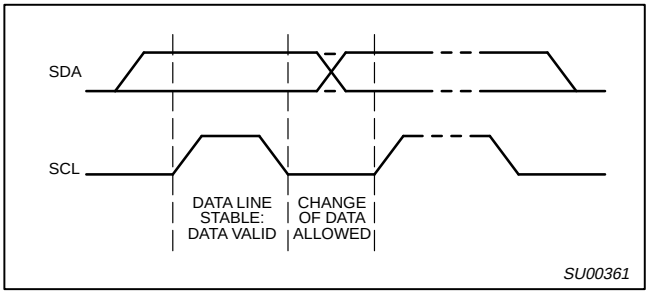

Figure 3. Bit Transfer on the I²C Bus

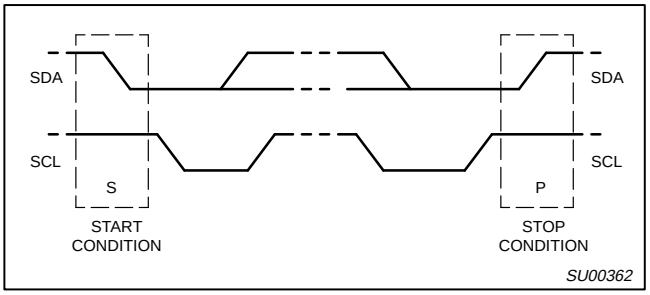

Figure 4. Start and Stop Conditions

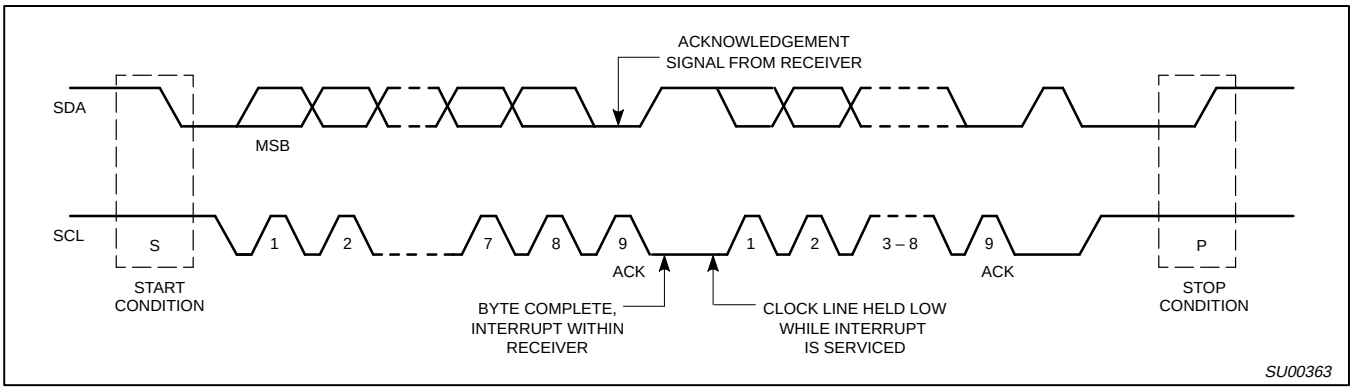

Figure 5. Data Transfer on the I²C Bus

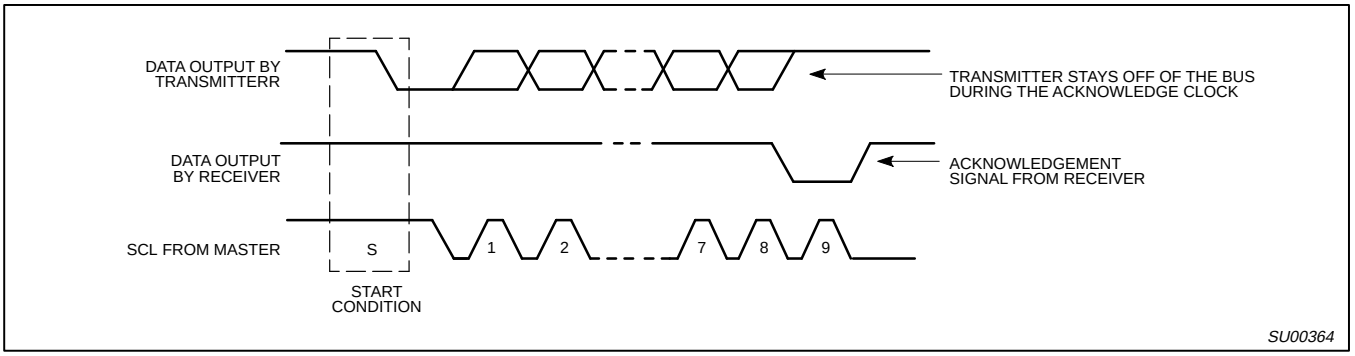

Figure 6. Acknowledge on the I²C Bus

A slave receiver must generate an acknowledge after the reception of each byte, and a master must generate one after the reception of each byte clocked out of the slave transmitter. If a receiving device cannot receive the data byte immediately, it can force the transmitter into a wait state by holding the clock line (SCL) LOW. When designing a system, it is necessary to take into account cases when acknowledge is not received. This happens, for example, when the addressed device is busy in a real time operation. In such a case the master, after an appropriate "time-out", should abort the transfer by generating a Stop condition, allowing other transfers to take place. These "other transfers" could be initiated by other masters in a multimaster system, or by this same master.

There are two exceptions to the "acknowledge after every byte" rule. The first occurs when a master is a receiver: it must signal an end of data to the transmitter by NOT signalling an acknowledge on the last byte that has been clocked out of the slave. The acknowledge related clock, generated by the master should still take place, but the SDA line will not be pulled down. In order to indicate that this is an active and intentional lack of acknowledgement, we shall term this special condition as a "negative acknowledge".

The second exception is that a slave will send a negative acknowledge when it can no longer accept additional data bytes. This occurs after an attempted transfer that cannot be accepted.

The bus design includes special provisions for interfacing to microprocessors which implement all of the I²C communications in software only—it is called "Slow Mode". When all of the devices on the network have built-in I²C hardware support, the Slow Mode is irrelevant.

Addressing and Transfer Formats

Each device on the bus has its own unique address. Before any data is transmitted on the bus, the master transmits on the bus the address of the slave to be accessed for this transaction. A well-behaved slave with a matching address, if it exists on the network, should of course acknowledge the master's addressing. The addressing is done by the first byte transmitted by the master after the Start condition.

An address on the network is seven bits long, appearing as the most significant bits of the address byte. The last bit is a direction (R/W) bit. A zero indicates that the master is transmitting (WRITE) and a one indicates that the master requests data (READ). A complete data transfer, comprised of an address byte indicating a WRITE and two data bytes is shown in Figure 7.

When an address is sent, each device in the system compares the first seven bits after the Start with its own address. If there is a match, the device will consider itself addressed by the master, and will send an acknowledge. The device could also determine if in this transaction it is assigned the role of a slave receiver or slave transmitter, depending on the R/W bit.

Each node of the I²C network has a unique seven bit address. The address of a microcontroller is of course fully programmable, while peripheral devices usually have fixed and programmable address portions. In addition to the "standard" addressing discussed here, the I²C bus protocol allows for "general call" addressing and interfacing to CBUS devices.

When the master is communicating with one device only, data transfers follow the format of Figure 7, where the R/W bit could indicate either direction. After completing the transfer and issuing a Stop condition, if a master would like to address some other device on the network, it could of course start another transaction, issuing a new Start.

Another way for a master to communicate with several different devices would be by using a "repeated start". After the last byte of the transaction was transferred, including its acknowledge (or negative acknowledge), the master issues another Start, followed by address byte and data—without effecting a Stop. The master may communicate with a number of different devices, combining READS and WRITES. After the last transfer takes place, the master issues a Stop and releases the bus. Possible data formats are demonstrated in Figure 8. Note that the repeated start allows for both change of a slave and a change of direction, without releasing the bus. We shall see later on that the change of direction feature can come in handy even when dealing with a single device.

In a single master system, the repeated start mechanism may be more efficient than terminating each transfer with a Stop and starting again. In a multimaster environment, the determination of which format is more efficient could be more complicated, as when a master is using repeated starts it occupies the bus for a long time and thus preventing other devices from initiating transfers.

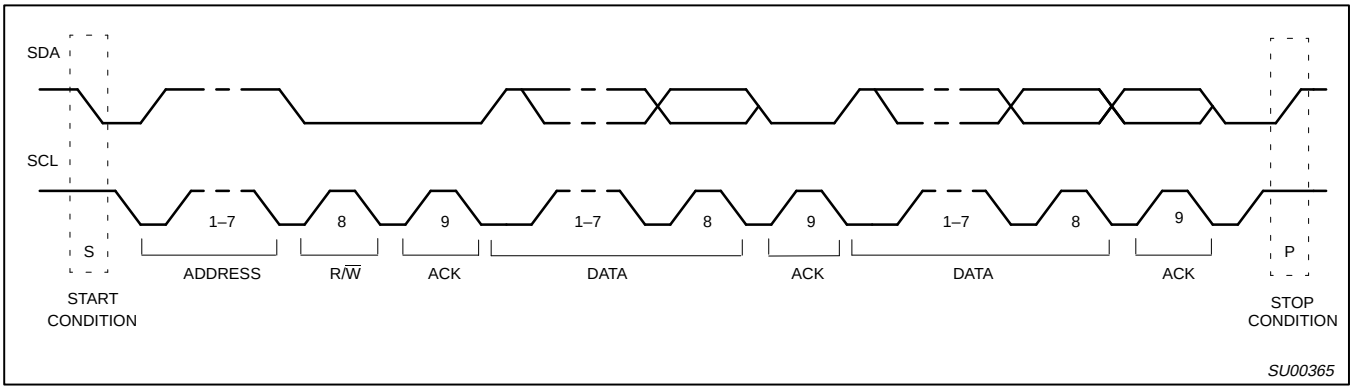

Figure 7. A Complete Data Transfer on the I²C-Bus

Using the 8XC751 microcontroller as an I²C bus master

AN422

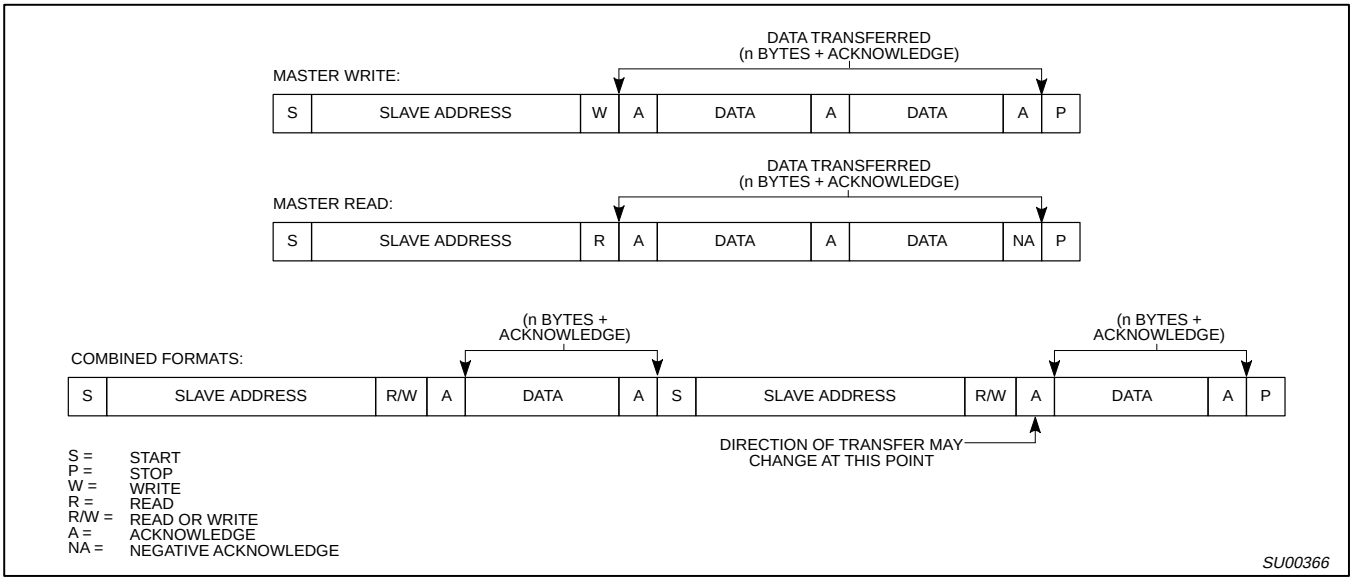

Figure 8. I²C Data Formats

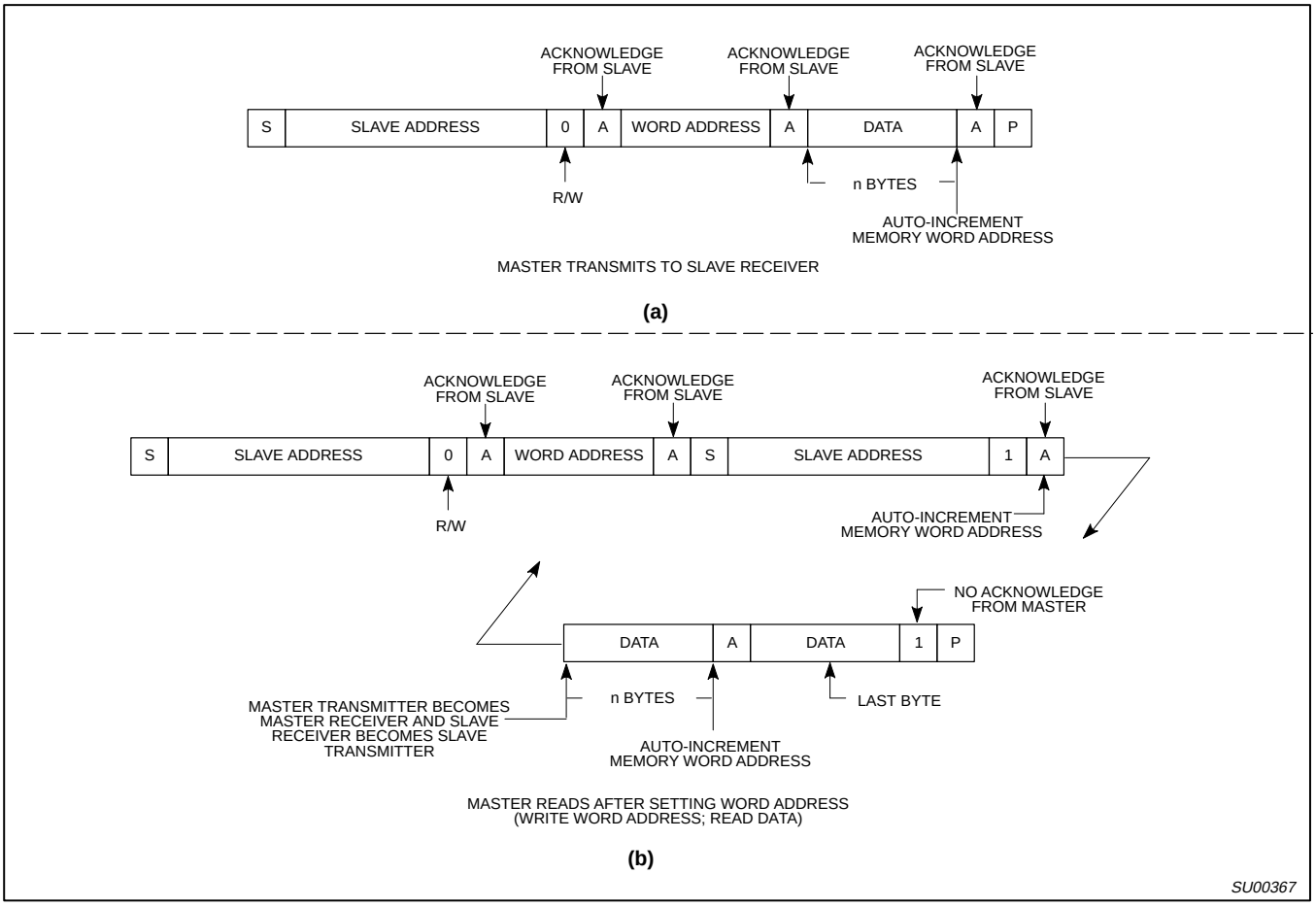

Figure 9. I²C Sub-Address Usage

Use of Sub-Addresses

For some ICs on the I²C bus, the device address alone is not sufficient for effective communications, and a mechanism for addressing the internals of the device is necessary. A typical example when we want to access a specific word inside the device is addressing memories, or a sequence of memory locations starting at a specific internal address.

A typical I²C memory device like the PCF8570 RAM contains a built-in word address register that is incremented automatically after each data byte which is a read or written data byte. When a master communicates with the PCF8570 it must send a sub-address in the byte following the slave address byte. This sub-address is the internal address of the word the master wants to access for a single byte transfer, or the beginning of a sequence of locations for a multi-byte transfer. A sub-address is an 8-bit byte, unlike the device address, it does not contain a direction (R/W) bit, and like any byte transferred on the bus it must be followed by an acknowledge.

A memory write cycle is shown in Figure 9(a). The Start is followed by a slave byte with the direction bit set to WRITE, a sub-address byte, a number of data bytes and a Stop signal. The sub-address is loaded into the word address memory, and the data bytes which follow will be written one after the other starting with the sub-address location, as the register is incremented automatically.

The memory read cycle (see Figure 9(b)) commences in a similar manner, with the master sending a slave address with the direction bit set to WRITE with a following sub-address. Then, in order to reverse the direction of the transfer, the master issues a repeated Start followed again by the memory device address, but this time with the direction bit set to READ. The data bytes starting at the internal sub-address will be clocked out of the device, each followed by a master-generated acknowledge. The last byte of the read cycle will be followed by a negative acknowledge, signalling the end of transfer. The cycle is terminated by a Stop signal.

8XC751 I²C HARDWARE

The on-chip I²C bus hardware support of the 8XC751 allows operation on the bus at full speed, and simplifies the software needed for effective communications on the network. The hardware activates and monitors the SDA and SCL lines, performs the necessary arbitration and framing errors checks, and takes care of clock stretching and synchronization. The hardware support includes a bus time-out timer, called Timer I. The hardware is synchronized to the software either through polled loops or interrupts.

Two of the port 0 pins are multi-functional. When the I²C is active, the pin associated with P0.0 functions as SCL, and the pin associated with P0.1 functions as SDA. These pins have an open drain output.

Two of the five 8XC751 interrupt sources may be used for I²C support. The I²C interrupt is enabled by the EI2 flag of the interrupt enable register, and its service routine should start at address 023h. An I²C interrupt is usually requested (if enabled) when a rising edge of SCL indicates a new data bit on the bus, or a special condition occurs: Start, Stop or arbitration loss. The interrupt is induced by the ATN flag—see below for the conditions for setting this flag. The Timer I overflow interrupt is enabled by the ETI flag, and the service routine starts at 01Bh.

The I²C port is controlled through three special function registers: I²C Control (I2CON), I²C Configuration (I2CFG), and I²C Data (I2DAT). The register addresses are shown in Table 1.

Although the following discussion of the hardware and register details is not complete, it should give a better understanding of the programming examples.

Timer I

In I²C applications, Timer I is dedicated to the port timing generation and bus monitoring. In non-I²C applications, it is available for use as a fixed time base.

In its port timing generation function, Timer I is used to generate SCL, the I²C clock. Timer I is clocked once per machine cycle (osc/12), so that the toggle rate of SCL will be some multiple of that rate. Because the 83C751 can be run over a wide range of oscillator frequencies, it is necessary to adjust SCL for the part's oscillator frequency. This allows the I²C bus to be used at its highest transfer rates independent of the oscillator frequency. SCL is adjusted by writing to two bits (CT0 and CT1) in the I2CFG special function register (see Table 2). The inverse of the values in CT0 and CT1 are loaded into the least significant two bit locations of Timer I every time the fourth bit of the timer is toggled. (A value is actually loaded into the least significant three bits, the third bit being 0 unless both CT0 and CT1 are programmed high and in that case the third bit is 1). SCL is then toggled every time the fourth bit of Timer I is toggled. For example: if CT1 = 0 and CT0 = 1 then the least significant three bits of Timer I would be preloaded with 2 (010 binary). Timer I would then count 3, 4, 5, 6, 7, 8 (6 counts or machine cycles). On 8, the fourth bit of Timer I will toggle, SCL will toggle and the 3 least significant bits will again be preloaded with the value 2 (010).

Table 1. I²C Special Function Register Addresses

REGISTER			BIT ADDRESS							
Name	Symbol	Address	MSB							LSB
I²C Control	I2CON	98	9F	9E	9D	9C	9B	9A	99	98
I²C Data	I2DAT	99	–	–	–	–	–	–	–	–
I²C Configuration	I2CFG	D8	DF	DE	DD	DC	DB	DA	D9	D8

Table 2. CT0, CT1 Timer I Settings

CT1 Values	CT0 Values	Timer I Counts	Oscillator Freq (MHz)
1	0	7	16
0	1	6	15, 14, 13
0	0	5	12, 11
1	1	4	10 or less

Timer I counts = f_{OSC} (MHz) x 0.39 (rounded up to next integer).

For the bus monitoring function, Timer I is used as a "watchdog timer" for bus hang-ups. It creates an interrupt when the SCL line stays in one state for an extended period of time while the bus is active (between a Start condition and a following Stop condition). SCL "stuck low" indicates a faulty master or slave. SCL "stuck high" may mean a faulty device, or that noise induced unto the I²C caused all masters to withdraw from the I²C arbitration.

The time-out interval of Timer I is fixed (cannot be set): it carries out and interrupts (if enabled) when about 1024 machine cycles have elapsed since a change on SCL within a frame. In other words, whenever I²C is active and Timer I is enabled, the falling edge of SCL will reset Timer I. If SCL is not toggled low for 1024 machine cycles, Timer I will overflow and cause an interrupt. (Note: we wrote "about 1024 machine cycles" although for the sake of accuracy—this number is affected by the setting of the CT0 and CT1 bits mentioned above and may vary by up to three machine cycles) The exact number of cycles for a time-out is not critical; what is important is that it indicates SCL is stuck.

In addition to the interrupt, upon Timer I overflow the I²C port hardware is reset. This is useful for multiple master systems in situations where a bus fault might cause the bus to hang-up due to a lack of software response. When this happens, SCL will be released, and I²C operation between other devices can continue.

I2CON Register

The I²C control register (I2CON) can be written to (see Figure 10). When writing to the I2CON register, one should use bit masks as demonstrated in the example program. Trying to clear or set the bits in the register using the bit addressing capabilities of the 8XC751 may lead to undesirable results. The reason is that a command like CLR reads the register, sets the bit and writes it back, and the write-back may affect other bits.

I2CFG Register

The configuration register (I2CFG) is a read/write register (see Figure 11).

I2DAT Register

The I²C data register (I2DAT) is a read/write register, where the MSB represents the data received or data to be sent. The other seven bits are read as 0 (see Figure 12).

Transmit Active State

The transmit active state—Xmit Active—is an internal state in the I²C interface that is affected by the I²C registers as explained above. The I²C interface will only drive the SDA line low when Xmit Active is set. Xmit Active is set by writing the I2DAT register, or by writing I2CON with XSTR = 1 or XSTP = 1. The ARL bit will be set to 1 only when Xmit Active is set—in such a case Xmit Active will be automatically reset upon arbitration loss. Xmit Active is cleared by writing 1 to CXA at I2CON register or by reading the I2DAT register.

I2CON READ	RDAT	ATN	DRDY	ARL	STR	STP	MASTER	——

RDAT	Received DATa bit. The value of SDA latched by the rising edge of SCL. Its contents is identical to RDAT in the I2DAT register. Reading the received data here allows doing so without clearing DRDY and releasing SCL.
ATN	An "ATteNtion" flag, set when any one of DRDY, ARL, STR or STP is set. This flag allows a single bit testing for terminating "wait loops", indicating a meaningful event on the bus. This flag also activates the I²C interrupt request.
DRDY	Data ReaDY flag. Set by a rising edge of SCL when I²C is active, except at an idle slave. This flag is cleared by reading or writing the I2DAT register, or by writing a 1 to CDR (at the same address, when I2CON is written).
ARL	ARbitration Loss flag. Indicates that this device lost arbitration while trying to take control of the bus.
STR	STaRt flag. Set when a Start condition is detected, except at an idle slave.
STP	SToP flag. Set when a Stop condition is detected, except at an idle slave.
MASTER	This flag is set when the controller is a bus master (or a potential master, prior to arbitration).

I2CON WRITE	CXA	IDLE	CDR	CARL	CSTR	CSTP	XSTR	XSTP

CXA	"Clear Xmit Active". Writing a 1 to CXA clears the internal transmit-active state.
IDLE	Setting this bit will cause a slave to enter idle mode and ignore the I²C bus until the next Start is detected. If the software sets the MASTRQ flag, the device may stop idling by turning into a master.
CDR	Clear Data Ready. Clears the DRDY flag.
CARL	Clear Arbitration Lost. Clears the ARL flag.
CSTR	Clear STaRt. Clears the STR flag.
CSTP	Clear STop. Clears the STP flag.
XSTR	"Xmit repeated STaRt". Writing a 1 to this bit causes the hardware to issue a Repeated Start signal. A side effect will be setting the internal Xmit Active state. This should be used only when the device is a master.
XSTP	"Xmit SToP". Issues a Stop condition. The Xmit active state is set.

SU00368

Figure 10. I2CON Register

SLAVEN	MASTRQ	CLRTI	TIRUN	—	—	CT1	CT0

SLAVEN Writing a 1 to this flag enables the slave functions of the I²C interface.

MASTRQ Request control of the bus as a master.

CLRTI Clear the Timer I interrupt flag. This bit is always read as 0.

TIRUN Writing a 1 will let Timer I run. When I2C is active, it will run only inside frames, and will be cleared by SCL transitions, Start and Stop. Writing a 0 will stop and clear the timer.

CT1, CT0 These bits should be programmed according to the frequency of the crystal oscillator used in the hardware. They determine the minimum high and low times for SCL, and are used to optimized performance at different oscillator speeds.

SU00369

Figure 11. I2CFG Register

I2DAT READ	RDAT	—	—	—	—	—	—	—

RDAT Received DATa bit, captured from SDA every rising edge of SCL. Reading I2CAT clears DRDY and the Xmit Active state. If it is necessary to read the data without affecting the flags, it can be read out of RDAT in the I2CON register.

I2DAT WRITE	XDAT	—	—	—	—	—	—	—

XDAT Xmit DATa bit. Writing XDAT determines the data for the next bit to be transmitted on the I²C bus. Writing I2DAT also clears DRDY and sets the Xmit Active state.

SU00370

Figure 12. I2DAT Register

PROGRAMMING EXAMPLE

The listing demonstrates communications routines for the 8XC751 as an I²C bus master in a single-master system.

The single-master system is less complicated than a multimaster environment. The programmer does not have to worry about switching between master and slave roles, or the consequences of an arbitration loss.

The I²C interrupt is not used, and therefore disabled. There is no need for frame Start interrupts, as this processor is the only bus master and all data transfers are initiated by it when the appropriate routines are called by the application. No one else generates frame Starts which could be an interrupt source in a multimaster system. Within the frames we monitor bus activity with a wait-loop which polls the ATN flag. As we expect the bus to operate in its full-speed mode, we can assume that only a small amount of time will be wasted in those loops, and the use of interrupts would be less efficient.

The 8XC751 has single-bit I²C hardware interface, where the registers may directly affect the levels on the bus and the software interacting with the register takes part in the protocol implementation. The hardware and the low-level routines dealing with the registers are tightly coupled. Therefore, one should take extra care if trying to modify these lower level routines.

The beginning of the program, at address 0, contains the reset vector, where the microcontroller begins executing code after a hardware reset. In this case, the code simply jumps to the main part of the program, which begins at the label ‘Reset’ near the end of the listing.

The main program is a simple demonstration of the I²C routines which comprise the balance of the listing. It first enables the Timer I interrupt, and sets up some sample data to be transmitted. Beginning at the label MainLoop, the program then proceeds to transmit one byte of data to a slave device at address 48 hexadecimal, using the routine titled ‘SendData’. In our demonstration hardware, this address corresponds to an 8-bit I/O port that drives eight monitor LEDs. The program then reads back one byte of data from the same port using the routine ‘RcvData’. The SendData and RcvData routines can send or receive multiple bytes of data, the number of which is determined by the variable ‘ByteCnt’.

Upon return from both SendData and RcvData, the program checks the system flag named ‘Retry’ to see if the transfer was completed correctly. If not, it loops back and attempts the same transfer again.

Next, the program sends four bytes of data to a 256-byte EEPROM device, an 8-pin part called the PCF8582. The routine ‘SendSub’ is used for this purpose. The EEPROM was located at address A0 hexadecimal on our board. This device uses the sub-addressing feature to select a starting location to address in the EEPROM array. When data is written to the EEPROM, the address is automatically incremented so that the data bytes are stored in consecutive locations.

Finally the program reads back four bytes of data from the EEPROM using the routine ‘RcvSub’. Calls to SendSub and RcvSub should also be followed by a test of the Retry flag to insure that all went according to plan.

This entire process is repeated indefinitely by jumping back to MainLoop.

Back at the beginning of the program, the next location after the reset vector is the Timer I interrupt service routine. The microcontroller will go to address 1B hexadecimal if Timer I overflows. This routine stops the timer, clears the timer interrupt, clears the pending interrupt so that other interrupts will be enabled, restores the stack pointer, and jumps to the 'Recover' routine to try to correct whatever stopped the I2C bus and allowed Timer I to overflow.

Next in the listing come the main I2C service routines. These are the routines SendData, RcvData, SendSub, and RcvSub that were called from the main program. Both of the send routines use the data area labeled 'XmtDat' as the transmit data buffer. In this sample program, four bytes were reserved for this area, but it could be larger or smaller depending on the application. The two receive routines use another four byte buffer labeled 'RcvDat' to store received data. All of these routines use the variables 'SlvAdr' and 'ByteCnt' to determine the slave address and the number of bytes to be sent or received, respectively. The SendSub and RcvSub routines use the variable 'SubAdr' as the sub-address to send to the slave device.

Following the main I2C service routines in the listing are the subroutines that are called by the main routines to deal intimately with the I2C hardware.

The 'SendAddr' subroutine requests mastership of the I2C bus and calls the routine 'XmitAddr' to complete sending the slave address. The bulk of the XmitAddr routine is shared with the 'XmitByte' subroutine which sends data bytes on the I2C bus. XmitByte is also used to send I2C sub-addresses. Both subroutines check for an acknowledge from the slave device after every byte is sent on the I2C bus.

The next subroutine 'RDAck' calls the 'RcvByte' routine to read in a byte of data. It then sends an acknowledge to the slave device. RDAck is used to receive all data except for the last byte of a receive data frame, where the acknowledge is omitted by the bus master. The RcvByte subroutine is called directly for the last byte of a frame.

The 'SendStop' subroutine causes a stop condition on the I2C, thus ending a frame. The 'RepStart' subroutine sends a repeated start condition on the I2C bus, to allow the master to start a new frame without first having to send an intervening stop.

The lower level subroutines deal directly with the hardware. The tight coupling between hardware and software is best demonstrated by the following explanations, relating to two cases in which the code is not self evident.

Sending the Address

When sending the address byte in the Send Addr subroutine, the first bit is written to I2DAT prior to the loop where the other seven bits are sent (SendAd2). The reason is that we need to clear the Start condition in order to release the SCL line, and this is done explicitly by the subsequent command. When SCL is released, the correct bit (MSB of address) must already be in I2DAT.

Capturing the Received Data

Typically, a program receiving data waits in a loop for ATN, and when detected, checks DRDY. If DRDY = 1 then there was a rising SCL, and the new data can be read from RDAT in I2CON or I2DAT. Reading or writing I2DAT clears DRDY, thus releasing SCL.

When reading the last bit in a byte, it should be read from I2CON, and not I2DAT (see the end of the RcvByte routine). This way the Data Ready (DRDY) flag is not cleared, and the low period on SCL is stretched. The reason for doing so is that upon reception of the last bit of a received byte the master must react with an acknowledge. In order to ensure that we "wait" with the acknowledge clock (release of SCL) until the acknowledge level is issued on SDA, the last bit is read out of I2CON and not I2DAT. SCL is stretched low until the acknowledge level is written into I2DAT by the software.

Bus Faults and Other Exceptions

Bus exceptions are detected either by Timer I time-out, or "illegal" logic states tested for and detected by the software. Upon Timer I time-out, a bus recovery is attempted by the Recover routine. The final section of the listing is this 'Recover' routine. Its job is to try to restore control of the I2C bus to the main program. First, the subroutine 'FixBus' is called. It checks to see if only the SDA line is 'stuck', and if so, tries to correct it by sending some extra clocks on the SCL line, and forcing a stop condition on the bus. If this does not work, another subroutine 'BusReset' is called. This generally happens when a severe bus error occurs, such as a shorted clock line. The philosophy used in this code is that the only chance of recovering from a severe error is to cause a reset of the I1C hardware by deliberately forcing Timer I to time out. This method allows recovery from a temporary short or other serious condition on the I2C bus.

```
I2CAPP          83C751 Single Master I2C Routines                09/07/89

                 1
                 2    ;*******************************************************
                 3    ;
                 4    ;   Sample I2C Single Master Routines for the 83C751
                 5
                 6    ;*******************************************************
                 7
                 8    $TITLE(83C751 Single Master I2C Routines)
                 9    $DATE(09/07/89)
                10    $MOD751
                11    $DEBUG
                12
                13
                14    ; Value definitions.
                15
  0002          16    CTVAL      EQU     02h      ;CT1, CT0 bit values for I2C.
                17
                18
                19    ; Masks for I2CFG bits.
                20
  0010          21    BTIR       EQU     10h      ;Mask for TIRUN bit.
  0040          22    BMRQ       EQU     40h      ;Mask for MASTRQ bit.
                23
                24
                25    ; Masks for I2CON bits.
                26
  0080          27    BCXA       EQU     80h      ;Mask for CXA bit.
  0040          28    BIDLE      EQU     40h      ;Mask for IDLE bit.
  0020          29    BCDR       EQU     20h      ;Mask for CDR bit.
  0010          30    BCARL      EQU     10h      ;Mask for CARL bit.
  0008          31    BCSTR      EQU     08h      ;Mask for CSTR bit.
  0004          32    BCSTP      EQU     04h      ;Mask for CSTP bit.
  0002          33    BXSTR      EQU     02h      ;Mask for XSTR bit.
  0001          34    BXSTP      EQU     01h      ;Mask for XSTP bit.
                35
                36
                37    ; RAM locations used by I2C routines.
                38
  0021          39    BitCnt     DATA    21h     ;I2C bit counter.
  0022          40    ByteCnt    DATA    22h
  0023          41    SlvAdr     DATA    23h     ;Address of active slave.
  0024          42    SubAdr     DATA    24h
                43
  0025          44    RcvDat     DATA    25h     ;I2C receive data buffer (4 bytes).
                45                                ; addresses 25h through 28h.
                46
  0029          47    XmtDat     DATA    29h     ;I2C transmit data buffer (4 bytes).
                48                                ;  addresses 29h through 2Ch.
                49
  002D          50    StackSave  DATA    2Dh     ;Saves stack addr for bus recovery.
                51
  0020          52    Flags      DATA    20h     ;I2C software status flags.
  0000          53    NoAck      BIT     Flags.0 ;Indicates missing acknowledge.
  0001          54    Fault      BIT     Flags.1 ;Indicates a bus fault of some kind.
  0002          55    Retry      BIT     Flags.2 ;Indicates that last I2C transmission
                56                                ; failed and should be repeated.
                57
  0080          58    SCL        BIT     P0.0    ;Port bit for I2C serial clock line.
  0081          59    SDA        BIT     P0.1     ;Port bit for I2C serial data line.
                60
```

```
                 61    ;*************************************************************
                 62    ;                                 Begin Code
                 63    ;*************************************************************
                 64
                 65    ; Reset and interrupt vectors.
                 66
0000 21E1        67              AJMP    Reset               ;Reset vector at address 0.
                 68
                 69
                 70    ; A timer I timeout usually indicates a 'hung' bus.
                 71
001B             72              ORG     1Bh                 ;Timer I (I2C timeout)
                                                             ; interrupt.
001B D2DD        73    TimerI:   SETB    CLRTI               ;Clear timer I interrupt.
001D C2DC        74              CLR     TIRUN
001F 1126        75              ACALL   ClrInt              ;Clear interrupt pending.
0021 852D81      76              MOV     SP,StackSave        ;Restore stack for return
                                                             ; to main.
0024 218A        77              AJMP    Recover             ;Attempt bus recovery.
0026 32          78    ClrInt:   RETI
                 79
                 80
                 81    ;*************************************************************
                 82    ;               Main Transmit and Receive Routines
                 83    ;*************************************************************
                 84
                 85    ; Send data byte(s) to slave.
                 86    ;  Enter with slave address in SlvAdr, data in XmtDat buffer,
                 87    ;   # of data bytes to send in ByteCnt.
                 88
0027 C200        89    SendData: CLR     NoAck               ;Clear error flags.
0029 C201        90              CLR     Fault
002B C202        91              CLR     Retry
002D 85812D      92              MOV     StackSave,SP        ;Save stack address
                                                               for bus fault.
0030 E523        93              MOV     A,SlvAdr            ;Get slave address.
0032 310C        94              ACALL   SendAddr            ;Get bus and send slave addr.
0034 200012      95              JB      NoAck,SDEX          ;Check for missing
                                                             ; acknowledge.
0037 200112      96              JB      Fault,SDatErr       ;Check for bus fault.
003A 7829        97              MOV     R0,#XmtDat          ;Set start of transmit
                                                             ; buffer.
                 98
003C E6          99    SDLoop:   MOV     A,@R0               ;Get data for slave.
003D 08          100             INC     R0
003E 3125        101             ACALL   XmitByte            ;Send data to slave.
0040 200006      102             JB      NoAck,SDEX          ;Check for missing
                                                             ; acknowledge.
0043 200106      103             JB      Fault,SDatErr       ;Check for bus fault.
0046 D522F3      104             DJNZ    ByteCnt,SDLoop
                 105
0049 3166        106    SDEX:    ACALL   SendStop            ;Send an I2C stop.
004B 22          107             RET
                 108
                 109
                 110    ; Handle a transmit bus fault.
                 111
004C 218A        112    SDatErr:  AJMP    Recover           ;Attempt bus recovery.
                 113
                 114
                       115    ; Receive data byte(s) from slave.
                       116    ; Enter with slave address in SlvAdr,
                              ;  # of data bytes requested in ByteCnt.
                       117    ;  Data returned in RcvDat buffer.
```

```
                    118
004E C200           119    RcvData:   CLR     NoAck             ;Clear error flags.
0050 C201           120               CLR     Fault
0052 C202           121               CLR     Retry
0054 85812D         122               MOV    StackSave,SP       ;Save stack address
                                                                ; for bus fault.
0057 E523           123               MOV     A,SlvAdr          ;Get slave address.
0059 D2E0           124               SETB    ACC.0             ;Aet bus read bit.
005B 310C           125               ACALL   SendAddr          ;Send slave address.
005D 200023         126               JB      NoAck,RDEX        ;Check for missing
                                                                ; acknowledge.
0060 200123         127               JB      Fault,RDatErr     ;Check for bus fault.
                    128
0063 7825           129               MOV     R0,#RcvDat        ;Set start of receive
                                                                ; buffer.
0065 D52202         130               DJNZ    ByteCnt,RDLoop    ;Check for count = 1
                                                                ; byte only.
0068 800A           131               SJMP    RDLast
                    132
006A 3143           133   RDLoop:     ACALL   RDAck             ;Get data and send
                                                                ; an acknowledge.
006C 200117         134               JB      Fault,RDatErr     ;Check for bus fault.
006F F6             135               MOV     @R0,A             ;Save data.
0070 08             136               INC     R0
0071 D522F6         137               DJNZ    ByteCnt,RDLoop    ;Repeat until last
                                                                ; byte.
                    138
0074 314F           139    RDLast:    ACALL   RcvByte           ;Get last data byte
                                                                ; from slave.
0076 20010D         140               JB      Fault,RDatErr     ;Check for bus
                                                                ; fault.
0079 F6             141               MOV     @R0,A             ;Save data.
                    142
007A 759980         143               MOV     I2DAT,#80h        ;Send negative
                                                                ; acknowledge.
007D 309EFD         144               JNB     ATN,$             ;Wait for NAK sent.
0080 309D03         145               JNB     DRDY,RDatErr      ;Check for bus
                                                                ; fault.
                    146
0083 3166           147    RDEX:      ACALL   SendStop          ;Send an I2C bus
                                                                ; stop.
0085 22             148               RET
                    149
                    150
                    151    ; Handle a receive bus fault.
                    152
0086 218A           153    RDatErr:   AJMP    Recover           ;Attempt bus recovery.
                    154
                    155
                    156    ; Send data byte(s) to slave with subaddress.
                    157    ; Enter with slave address in ACC, subaddress in
                           ; SubAdr, # of bytes to send in ByteCnt,
                    158    ; data in XmtDat buffer.
                    159
0088 C200           160    SendSub:   CLR     NoAck              ;Clear error flags.
008A C201           161               CLR     Fault
008C C202           162               CLR     Retry
008E 85812D         163               MOV     StackSave,SP    ;Save stack address
                                                                ; for bus fault.
0091 E523           164               MOV     A,SlvAdr        ;Get slave address.
0093 310C           165               ACALL   SendAddr        ;Get bus and send
                                                                ; slave address.
```

```
0095 20001C          166              JB      NoAck,SSEX      ;Check for missing
                                                              ; acknowledge.
0098 20011C          167              JB      Fault,SSubErr   ; Check for bus
                                                              ; fault.
                     168
009B E524            169              MOV     A,SubAdr        ;Get slave subaddress.
009D 3125            170              ACALL   XmitByte        ;Send subaddress.
009F 200012          171              JB      NoAck,SSEX      ;Check for missing
                                                              ; acknowledge.
00A2 200112          172              JB      Fault,SSubErr   ;Check for bus fault.
00A5 7829            173              MOV     R0,#XmtDat      ;Set start of
                                                              ; transmit buffer.
                     174
00A7 E6              175    SSLoop:   MOV     A,@R0           ;Get data for slave.
00A8 08              176              INC     R0
00A9 3125            177              ACALL   XmitByte        ;Send data to slave.
00AB 200006          178              JB      NoAck,SSEX      ;Check for missing
                                                              ; acknowledge.
00AE 200106          179              JB      Fault,SSubErr   ;Check for bus fault.
00B1 D522F3          180              DJNZ    ByteCnt,SSLoop
                     181
00B4 3166            182    SSEX:     ACALL   SendStop        ;Send an I2C stop.
00B6 22              183              RET
                     184
                     185
                     186    ; Handle a transmit bus fault.
                     187
00B7 218A            188    SSubErr:  AJMP    Recover         ;Attempt bus recovery.
                     189
                     190
                     191    ; Receive data byte(s) from slave with subaddress.
                     192    ; Enter with slave address in SlvAdr, subaddress in SubAdr,
                            ; # of data bytes  requested in ByteCnt.
                     193    ; Data returned in RcvDat buffer.
                     194
00B9 C200            195    RcvSub:   CLR     NoAck           ;Clear error flags.
00BB C201            196              CLR     Fault
00BD C202            197              CLR     Retry
00BF 85812D          198              MOV     StackSave,SP    ;Save stack address
                                                              ; for bus fault.
00C2 E523            199              MOV     A,SlvAdr        ;Get slave address.
00C4 310C            200              ACALL   SendAddr        ;Send slave address.
00C6 20003E          201              JB      NoAck,RSEX      ;Check for missing
                                                              ; acknowledge.
00C9 20013E          202              JB      Fault,RSubErr   ;Check for bus fault.
                     203
00CC E524            204              MOV     A,SubAdr        ;Get slave subaddress.
00CE 3125            205              ACALL   XmitByte        ;Send subaddress.
00D0 200034          206              JB      NoAck,RSEX      ;Check for missing
                                                              ; acknowledge.
00D3 200134          207              JB      Fault,RSubErr   ;Check for bus fault.
                     208
00D6 317A            209              ACALL   RepStart        ;Send repeated start.
00D8 20012F          210              JB      Fault,RSubErr   ;Check for bus fault.
00DB E523            211              MOV     A,SlvAdr        ;Get slave address.
00DD D2E0            212              SETB    ACC.0           ;Set bus read bit.
00DF 3115            213              ACALL   SendAd2         ;Send slave address.
00E1 200023          214              JB      NoAck,RSEX      ;Check for missing
                                                              ; acknowledge.
00E4 200123          215              JB      Fault,RSubErr   ;Check for bus fault.
                     216
00E7 7825            217              MOV     R0,#RcvDat      ;Set start of
                                                              ; receive buffer.
00E9 D52202          218              DJNZ    ByteCnt,RSLoop  ;Check for count = 1
                                                              ; byte only.
```

```
00EC 800A           219             SJMP    RSLast
                    220
00EE 3143           221   RSLoop:   ACALL   RDAck          ;Get data and send
                                                           ; an acknowledge.
00F0 200117         222             JB      Fault,RSubErr  ;Check for bus fault.
00F3 F6             223             MOV     @R0,A          ;Save data.
00F4 08             224             INC     R0
00F5 D522F6         225             DJNZ    ByteCnt,RSLoop ;Repeat until last byte.
                    226
00F8 314F           227   RSLast:   ACALL   RcvByte        ;Get last data byte
                                                           ; from slave.
00FA 20010D         228             JB      Fault,RSubErr  ;Check for bus fault.
00FD F6             229             MOV     @R0,A          ;Save data.
                    230
00FE 759980         231             MOV     I2DAT,#80h     ;Send negative
                                                           ; acknowledge.
0101 309EFD         232             JNB     ATN,$          ;Wait for NAK sent.
0104 309D03         233             JNB     DRDY,RSubErr   ;Check for bus fault.
                    234
0107 3166           235   RSEX:     ACALL   SendStop       ;Send an I2C bus stop.
0109 22             236             RET
                    237
                    238
                    239   ; Handle a receive bus fault.
                    240
010A 218A           241   RSubErr:  AJMP    Recover        ;Attempt bus recovery.
                    242
                    243
                    244   ;*********************************************************
                    245   ;                    Subroutines
                    246   ;*********************************************************
                    247
                    248   ; Send address byte.
                    249   ;  Enter with address in ACC.
                    250
010C 75D852         251   SendAddr: MOV     I2CFG,#BMRQ+BTIR+CTVAL   ;Request I2C bus.
010F 309EFD         252             JNB     ATN,$                    ;Wait for bus
                                                                     ; granted.
0112 309908         253             JNB     Master,SAErr             ;Should have
                                                                     ; become the bus
                                                                     ; master.
0115 F599           254   SendAd2:  MOV     I2DAT,A                  ;Send first bit,
                                                                     ; clears DRDY.
0117 75981C         255             MOV     I2CON,#BCARL+BCSTR+BCSTP ;Clear start,
                                                                     ; releases SCL.
011A 3120           256             ACALL   XmitAddr                 ;Finish sending
                                                                     ; address.


011C 22             257             RET
                    258
011D D201           259   SAErr:    SETB    Fault                    ;Return bus fault
                                                                     ; status.
011F 22             260             RET
                    261
                    262
                    263   ; Byte transmit routine.
                    264   ;   Enter with data in ACC.
                    265   ;   XmitByte : transmits 8 bits.
                    266   ;   XmitAddr : transmits 7 bits (for address only).
                    267
0120 752108         268   XmitAddr: MOV     BitCnt,#8          ;Set 7 bits of
                                                               ; address count.
0123 8005           269             SJMP    XmBit2
                    270
```

```
0125 752108          271    XmitByte:  MOV     BitCnt,#8           ;Set 8 bits of data
                                                                   ; count.
0128 F599            272    XmBit:     MOV     I2DAT,A             ;Send this bit.
012A 23              273    XmBit2:    RL      A                   ;Get next bit.
012B 309EFD          274               JNB     ATN,$               ;Wait for bit sent.
012E 309D0F          275               JNB     DRDY,XMErr          ;Should be data ready.
0131 D521F4          276               DJNZ    BitCnt,XmBit        ;Repeat until all bits sent.
0134 7598A0          277               MOV     I2CON,#BCDR+BCXA    ;Switch to
                                                                   ; receive mode.
0137 309EFD          278               JNB     ATN,$               ;Wait for acknowledge
                                                                   ; bit.
013A 309F02          279               JNB     RDAT,XMBX           ;Was there an ack?
013D D200            280               SETB    NoAck               ;Return no acknowledge
                                                                   ; status.
013F 22              281    XMBX:      RET
                     282
0140 D201            283    XMErr:     SETB    Fault               ;Return bus fault
                                                                   ; status.
0142 22              284               RET
                     285
                     286
                     287    ; Byte receive routines.
                     288    ;   RDAck   : receives a byte of data, then sends
                            ;   an acknowledge.
                     289    ;   RcvByte : receives a byte of data.
                     290    ;   Data returned in ACC.
                     291
0143 314F            292    RDAck:     ACALL   RcvByte             ;Receive a data byte.
0145 759900          293               MOV     I2DAT,#0            ;Send receive
                                                                   ; acknowledge.
0148 309EFD          294               JNB     ATN,$               ;Wait for acknowledge
                                                                   ; sent.
014B 309D15          295               JNB     DRDY,RdErr          ;Check for bus fault.
014E 22              296               RET
                     297
014F 752108          298    RcvByte:   MOV     BitCnt,#8           ;Set bit count.
0152 E4              299               CLR     A                   ;Init received byte
                                                                   ; to 0.
0153 4599            300    RBit:      ORL     A,I2DAT             ;Get bit, clear ATN.
0155 23              301               RL      A                   ;Shift data.
0156 309EFD          302               JNB     ATN,$               ;Wait for next bit.
0159 309D07          303               JNB     DRDY,RdErr          ;Should be data ready.
015C D521F4          304               DJNZ    BitCnt,RBit         ;Repeat until 7 bits
                                                                   ; are in.
015F A29F            305               MOV     C,RDAT              ;Get last bit, don't
                                                                   ; clear ATN.
0161 33              306               RLC     A                   ;Form full data byte.
0162 22              307               RET
                     308
0163 D201            309    RdErr:     SETB    Fault               ;Return bus fault status.
0165 22              310               RET
                     311
                     312
                     313    ; I2C stop routine.
                     314
0166 C2DE            315    SendStop:  CLR     MASTRQ              ;Release bus
                                                                   ; mastership.
0168 759821          316               MOV     I2CON,#BCDR+BXSTP   ;Generate a bus stop.
016B 309EFD          317               JNB     ATN,$               ;Wait for atn.
016E 759820          318               MOV     I2CON,#BCDR         ;Clear data ready.
0171 309EFD          319               JNB     ATN,$               ;Wait for stop sent.
0174 759894          320               MOV     I2CON,#BCARL+BCSTP+BCXA ;Clear I2C bus.
0177 C2DC            321               CLR     TIRUN               ;Stop timer I.
0179 22              322               RET
                     323
```

```
                    324
                    325    ; I2C repeated start routine.
                    326    ;   Enter with address in ACC.
                    327
017A 759822         328    RepStart:  MOV     I2CON,#BCDR+BXSTR   ;Send repeated start.
017D 309EFD         329               JNB     ATN,$               ;Wait for ATN.
0180 759820         330               MOV     I2CON,#BCDR         ;Clear data ready.
0183 309EFD         331               JNB     ATN,$               ;Wait for repeated
                                                                  ; start sent.
0189 22             333               RET
                    334
                    335
                    336    ; Bus fault recovery routine.
                    337
018A 31A4           338    Recover:   ACALL   FixBus              ;See if bus is dead or
                                                                  ; can be 'fixed'.
018C 400D           339               JC      BusReset            ;If not 'fixed', try
                                                                  ; extreme measures.
018E D202           340               SETB    Retry               ;If bus OK, return to
                                                                  ; main routine.
0190 C201           341               CLR     Fault
0192 C200           342               CLR     NoAck
0194 D2DD           343               SETB    CLRTI
0196 D2DC           344               SETB    TIRUN               ;Enable timer I.
0198 D2AB           345               SETB    ETI                 ;Turn on timer I
                                                                  ; interrupts.
019A 22             346               RET
                    347
                    348    ;This routine tries a more extreme method of bus recovery.
                    349    ; This is used if SCL or SDA are stuck and cannot
                           ; otherwise be freed.
                    350    ; (will return to the Recover routine when Timer I times out)
                    351
019B C2DE           352    BusReset:  CLR     MASTRQ              ;Release bus.
019D 7598BC         353               MOV     I2CON,#0BCh         ;Clear all I2C flags.
01A0 D2DC           354               SETB    TIRUN
01A2 80FE           355               SJMP    $                   ;Wait for timer I
                                                                  ; timeout (this will re-
                    356                                           ; set the I2C hardware).
                    357
                    358
                    359    ; This routine attempts to regain control of the I2C
                           ;                bus after a bus fault.

                    360    ;  Returns carry clear if successful, carry set if failed.
                    361
01A4 C2DE           362    FixBus:    CLR     MastRQ             ;Turn off I2C functions.
01A6 D3             363               SETB    C
01A7 D280           364               SETB    SCL                ;Insure I/O port is not
                                                                 ; locking I2C.
01A9 D281           365               SETB    SDA
01AB 308029         366               JNB     SCL,FixBusEx       ;If SCL is low, bus
                                                                 ; cannot be 'fixed'.
01AE 208113         367               JB      SDA,RStop          ;If SCL & SDA are high,
                                                                 ; force a stop.
01B1 752109         368               MOV     BitCnt,#9          ;Set max # of tries to
                                                                 ; clear bus.
01B4 C280           369    ChekLoop:  CLR     SCL                ;Force an I2C clock.
01B6 31D8           370               ACALL   SDelay
01B8 208109         371               JB      SDA,RStop          ;Did it work?
01BB D280           372               SETB    SCL
01BD 31D8           373               ACALL   SDelay
01BF D521F2         374               DJNZ    BitCnt,ChekLoop    ;Repeat clocks until
                                                                 ; either SDA clears or
                    375                                          ; we run out of tries.
```

```
01C2 8013          376             SJMP    FixBusEx          ;Failed to fix bus by
                                                             ; this method.
                   377
01C4 C281          378   RStop:    CLR     SDA               ;Try forcing a stop
                                                             ; since SCL & SDA
01C6 31D8          379             ACALL   SDelay            ; are both high.
01C8 D280          380             SETB    SCL
01CA 31D8          381             ACALL   SDelay
01CC D281          382             SETB    SDA
01CE 31D8          383             ACALL   SDelay
01D0 308004        384             JNB     SCL,FixBusEx      ;Are SCL & SDA still
                                                             ; high? If so, assume bus
01D3 308101        385             JNB     SDA,FixBusEx      ; is now OK, and return
01D6 C3            386             CLR     C                 ; with carry cleared.
01D7 22            387   FixBusEx: RET
                   388
                   389
                   390   ; Short delay routine (10 machine cycles).
                   391
01D8 00            392   SDelay:   NOP
01D9 00            393             NOP
01DA 00            394             NOP
01DB 00            395             NOP
01DC 00            396             NOP
01DD 00            397             NOP
01DE 00            398             NOP
01DF 00            399             NOP
01E0 22            400             RET
                   401
                   402   ;*************************************************************
                   404   ;                     Main Program
                   405   ;*************************************************************
                   406
01E1 758107        407   Reset:    MOV     SP,#07h          ;Set stack location.
01E4 D2AB          408             SETB    ETI              ;Enable timer I interrupts.
01E6 D2AF          409             SETB    EA               ;Enable global interrupts.
01E8 75290B        410             MOV     XmtDat,#11       ;Set up transmit data.
01EB 752A16        411             MOV     XmtDat+1,#22     ;Set up transmit data.
01EE 752B2C        412             MOV     XmtDat+2,#44     ;Set up transmit data.
01F1 752C58        413             MOV     XmtDat+3,#88     ;Set up transmit data.
01F4 752500        414             MOV     RcvDat,#0       ;Clear receive data.
01F7 752600        415             MOV     RcvDat+1,#0     ;Clear receive data.
01FA 752700        416             MOV     RcvDat+2,#0     ;Clear receive data.
01FD 752800        417             MOV     RcvDat+3,#0     ;Clear receive data.
                   418
0200 752348        419   MainLoop: MOV     SlvAdr,#48h     ;Set slave address
                                                             ; (8-bit I/O port).
0203 752201        420             MOV     ByteCnt,#1      ;Set up byte count.
0206 1127          421             ACALL   SendData        ;Send data to slave.
0208 2002F5        422             JB      Retry,MainLoop
                   423
020B 752201        424   ML2:      MOV     ByteCnt,#1      ;Set up byte count.
020E 114E          425             ACALL   RcvData         ;Read data from slave.
0210 2002F8        426             JB      Retry,ML2
                   427
0213 7523A0        428   SL1:      MOV     SlvAdr,#0A0h    ;Set slave address
                                                             ; (RAM chip).
0216 752400        429             MOV     SubAdr,#0h      ;Set slave subaddress.
0219 752204        430             MOV     ByteCnt,#4      ;Set up byte count.
021C 1188          431             ACALL   SendSub
021E 2002F2        432             JB      Retry,SL1
                   433
```

```
0221 752204          434    SL2:       MOV     ByteCnt,#4     ;Set up byte count.
0224 11B9            435               ACALL   RcvSub
0226 2002F8          436               JB      Retry,SL2
                     437
0229 0529            438               INC     XmtDat
022B 052A            439               INC     XmtDat+1
022D 052B            440               INC     XmtDat+2
022F 052C            441               INC     XmtDat+3
0231 80CD            442               SJMP    MainLoop       ;Do it all again.
                     443
                     444               ENDASSEMBLY COMPLETE, 0 ERRORS FOUND

I2CAPP          83C751 Single Master I2C Routines
ACC. . . . . . . . . . . . . . . . .  D ADDR  00E0H  PREDEFINED
ATN. . . . . . . . . . . . . . . . .  B ADDR  009EH  PREDEFINED
BCARL. . . . . . . . . . . . . . . .    NUMB  0010H
BCDR . . . . . . . . . . . . . . . .    NUMB  0020H
BCSTP. . . . . . . . . . . . . . . .    NUMB  0004H
BCSTR. . . . . . . . . . . . . . . .    NUMB  0008H
BCXA . . . . . . . . . . . . . . . .    NUMB  0080H
BIDLE. . . . . . . . . . . . . . . .    NUMB  0040H  NOT USED
BITCNT . . . . . . . . . . . . . . .  D ADDR  0021H
BMRQ . . . . . . . . . . . . . . . .    NUMB  0040H
BTIR . . . . . . . . . . . . . . . .    NUMB  0010H
BUSRESET . . . . . . . . . . . . . .  C ADDR  019BH
BXSTP. . . . . . . . . . . . . . . .    NUMB  0001H
BXSTR. . . . . . . . . . . . . . . .    NUMB  0002H
BYTECNT. . . . . . . . . . . . . . .  D ADDR  0022H
CHEKLOOP . . . . . . . . . . . . . .  C ADDR  01B4H
CLRINT . . . . . . . . . . . . . . .  C ADDR  0026H
CLRTI. . . . . . . . . . . . . . . .  B ADDR  00DDH  PREDEFINED
CTVAL. . . . . . . . . . . . . . . .    NUMB  0002H
DRDY . . . . . . . . . . . . . . . .  B ADDR  009DH  PREDEFINED
EA . . . . . . . . . . . . . . . . .  B ADDR  00AFH  PREDEFINED
ETI. . . . . . . . . . . . . . . . .  B ADDR  00ABH  PREDEFINED
FAULT. . . . . . . . . . . . . . . .  B ADDR  0001H
FIXBUS . . . . . . . . . . . . . . .  C ADDR  01A4H
FIXBUSEX . . . . . . . . . . . . . .  C ADDR  01D7H
FLAGS. . . . . . . . . . . . . . . .  D ADDR  0020H
I2CFG. . . . . . . . . . . . . . . .  D ADDR  00D8H  PREDEFINED
I2CON. . . . . . . . . . . . . . . .  D ADDR  0098H  PREDEFINED
I2DAT. . . . . . . . . . . . . . . .  D ADDR  0099H  PREDEFINED
MAINLOOP . . . . . . . . . . . . . .  C ADDR  0200H
MASTER . . . . . . . . . . . . . . .  B ADDR  0099H  PREDEFINED
MASTRQ . . . . . . . . . . . . . . .  B ADDR  00DEH  PREDEFINED
ML2. . . . . . . . . . . . . . . . .  C ADDR  020BH
NOACK. . . . . . . . . . . . . . . .  B ADDR  0000H
P0 . . . . . . . . . . . . . . . . .  D ADDR  0080H  PREDEFINED
RBIT . . . . . . . . . . . . . . . .  C ADDR  0153H
RCVBYTE. . . . . . . . . . . . . . .  C ADDR  014FH
RCVDAT . . . . . . . . . . . . . . .  D ADDR  0025H
RCVDATA. . . . . . . . . . . . . . .  C ADDR  004EH
RCVSUB . . . . . . . . . . . . . . .  C ADDR  00B9H
RDACK. . . . . . . . . . . . . . . .  C ADDR  0143H
RDAT . . . . . . . . . . . . . . . .  B ADDR  009FH  PREDEFINED
RDATERR. . . . . . . . . . . . . . .  C ADDR  0086H
RDERR. . . . . . . . . . . . . . . .  C ADDR  0163H
RDEX . . . . . . . . . . . . . . . .  C ADDR  0083H
RDLAST . . . . . . . . . . . . . . .  C ADDR  0074H
RDLOOP . . . . . . . . . . . . . . .  C ADDR  006AH
RECOVER. . . . . . . . . . . . . . .  C ADDR  018AH
REPSTART . . . . . . . . . . . . . .  C ADDR  017AH
RESET. . . . . . . . . . . . . . . .  C ADDR  01E1H
RETRY. . . . . . . . . . . . . . . .  B ADDR  0002H
RSEX . . . . . . . . . . . . . . . .  C ADDR  0107H
```

```
RSLAST . . . . . .  C ADDR  00F8H
RSLOOP . . . . . .  C ADDR  00EEH
RSTOP. . . . . .  C ADDR  01C4H
RSUBERR. . . . . .  C ADDR  010AH
SAERR. . . . . .  C ADDR  011DH
SCL. . . . . .  B ADDR  0080H
SDA. . . . . .  B ADDR  0081H
SDATERR. . . . . .  C ADDR  004CH
SDELAY . . . . . .  C ADDR  01D8H
SDEX . . . . . .  C ADDR  0049H
SDLOOP . . . . . .  C ADDR  003CH
SENDAD2. . . . . .  C ADDR  0115H
SENDADDR . . . . . .  C ADDR  010CH
SENDDATA . . . . . .  C ADDR  0027H
SENDSTOP . . . . . .  C ADDR  0166H
SENDSUB. . . . . .  C ADDR  0088H
SL1. . . . . .  C ADDR  0213H
SL2. . . . . .  C ADDR  0221H
SLVADR . . . . . .  D ADDR  0023H
SP . . . . . .  D ADDR  0081H  PREDEFINED
SSEX . . . . . .  C ADDR  00B4H
SSLOOP . . . . . .  C ADDR  00A7H
SSUBERR. . . . . .  C ADDR  00B7H
STACKSAVE. . . . . .  D ADDR  002DH
SUBADR . . . . . .  D ADDR  0024H
TIMERI . . . . . .  C ADDR  001BH  NOT USED
TIRUN. . . . . .  B ADDR  00DCH  PREDEFINED
XMBIT. . . . . .  C ADDR  0128H
XMBIT2 . . . . . .  C ADDR  012AH
XMBX . . . . . .  C ADDR  013FH
XMERR. . . . . .  C ADDR  0140H
XMITADDR . . . . . .  C ADDR  0120H
XMITBYTE . . . . . .  C ADDR  0125H
XMTDAT . . . . . .  D ADDR  0029H
```